BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

FUNCTIONAL MEMORY AS A GENERAL PURPOSE SYSTEMS TECHNOLOGY

by

M. Flinders, P. L. Gardner, R. J. Llewelyn, J. F. Minshull

IBM United Kingdom Laboratories Limited.

ABSTRACT

Large Scale Integration (LSI) offers the equipment designer a dramatic reduction in circuit cost, at the expense of changeability during development and in the field. The semiconductor manufacturer is faced with an explosion in the number of chip types required and he becomes involved in a much more complex interface with the system designer.

In order to overcome these problems this paper proposes a Functional Memory module, based on a loadable array of three state associative cells, as a general purpose system technology.

Logical operations in Functional Memory are performed using table look-up methods. The paper describes a feasible FM module and the way in which tables for various logical operations can be designed. Modules may be grouped together to form a system data flow, and an example is given.

The use of this technology introduces a number of checking problems not normally encountered in conventional logic. Solutions to these problems have been devised which link effective fault detection and location with a potential for self-repair.

The paper concludes that this type of associative array logic can exploit the potential of LSI while eliminating some of the problems of today's conventional technology.

Functional Memory technology capitalises on the high circuit densities offered by LSI and is more versatile than current logic technologies. It allows functions to be changed easily and avoids the part number explosion.

ACKNOWLEDGEMENTS

The authors wish to acknowledge ideas contributed by numerous colleagues and to thank the management of International Business Machines Corporation for their permission to publish this paper.

INTRODUCTION

The advent of Large Scale Integration (LSI) promises significant reductions in the cost of electronic circuits. However, for the designer of complex digital systems it introduces a range of problems for which existing solutions are, in most cases, inadequate. He is faced with levels of integration well suited to the implementation of regular storage arrays, but not to the implementation of the multitude of variations encountered during the logical design of a digital system.

The feature of Large Scale Integration which causes the problems is the move towards a large number of circuits on a semiconductor chip. Past levels of technology have enabled the logic designer to use the building block principle, and construct his designs from a number of similar low level packages at card or module level. Variations in design, in this instance, are usually achieved by personalising the interconnections between standard packages, and in this way a range of units can employ a standard set of packages. Furthermore, when design errors and alterations in system specification occur, the system personality is at a level where it can easily be changed.

LSI is typically defined as putting more than 100 logic gates on a chip. This means that the logical function which the chip now carries is so complex that multiple usage of a particular chip design within a system is rare. It implies many different chip types as a result of the high level of personalisation of the chip. Moreover, much of the system personality is now in the form of metallised patterns on the surface of the chip and it cannot therefore be modified in situ. For this reason the introduction of system modifications involves long delays.

The low level package implemented in LSI should retain an acceptably low connector to circuit ratio. Its form should be independent of specific applications as this will enable chip design to continue without reference to the system design cycle. Changes in the user's designs should not require physical alterations of the package at a chip or module level.

Conventional logic methods used with LSI do not achieve these objectives. Alternative ways of exploiting LSI must be found if progress in logic organisations is not to be hindered. The regular store type structure, well suited to large scale integration techniques, provides a pointer to the logic media that will be most readily available in the future. The conventional binary addressed store may be made to perform logic, but is not well suited to this role. The associative array provides a regular structure in which data manipulation is more effectively performed. Previously described methods for processing the data held in an associative array provide a basis from which to work.

Some regularly structured parts of a digital system could possibly be translated directly into LSI (e.g. registers and arithmetic units). Areas least suited to high levels of integration are those having no regular structure, such as those performing machine control. The experimental logic element described has been proposed with this problem in mind.

The techniques described are those of processing data using associative storage. This data is passed against control and logic tables held in the associative arrays. This allows the array to be treated in a way which is analagous to that of the discrete logical element within current system organisations. It results in a wide variety of potential implementations, the efficiencies of which

are normally related to the logic designer's skill.

Because the logic of the system is represented in part by the contents of the arrays the efficiency with which this is stored, and used, can have a marked effect on the cost of implementing any given unit. The experimental associative cell employed in the logic array has three states. This type of cell was originally proposed by B. T. McKeever[1] for use with a cryogenic technology. It allows a considerable degree of table minimisation and thus helps to achieve an economical result.

This paper seeks to demonstrate how the logic designer can specify a system design in terms of tables held in interconnected stores.

1. CELL, ARRAY AND BOM

The element of array logic to be described is termed the Functional Memory Basic Operating Module (or FM BOM). The FM BOM contains a writable associative array which replaces the discrete logic element as the medium within which the logic of the system is defined. The array is supported by a number of additional components included within the FM BOM. They are the mask store and mask register, input-output register(s) and function controls. A general layout for the FM BOM is shown in Figure 1.

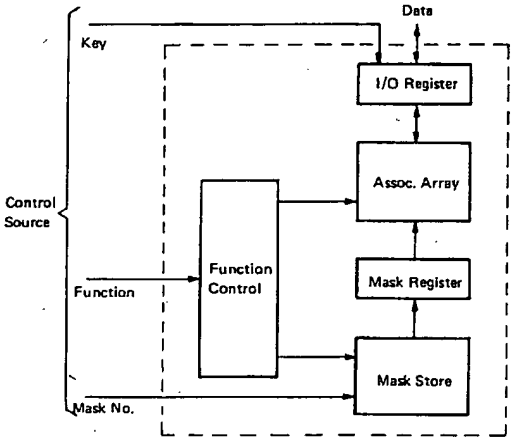

Figure 1. FM BOM

The basic cycle of an FM BOM is divided into two phases: these are the Select phase and the Read/Write phase as shown in Figure 2.

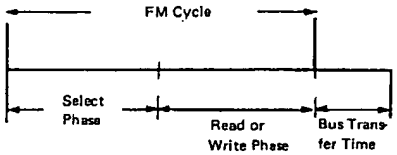

Figure 2. FM Cycle

The type of cell used in the array is shown in Figure 3. The cell has four stable states. Of these, three are normally given meaning in the implementation of the system logic. The cell consists of two similar bistables. Each bistable has a single emitter, which is held at a constant potential, and a dual emitter, by which it is linked to the array. The dual emitters are connected individually to bit and word lines. By suitable manipulation of the potential of these lines, the transistor current can be switched from one line to the other. The path into which the current is switched is dependent on the relative potentials of the bit and word lines. The two halves of the cell are connected to separate bit lines which are termed BIT 0 and BIT 1. The word line is shared. The conventions adopted for the states of the cell shown in Figure 3 are;

0 = T1, T3 on
1 = T2, T4 on
X = T2, T3 on

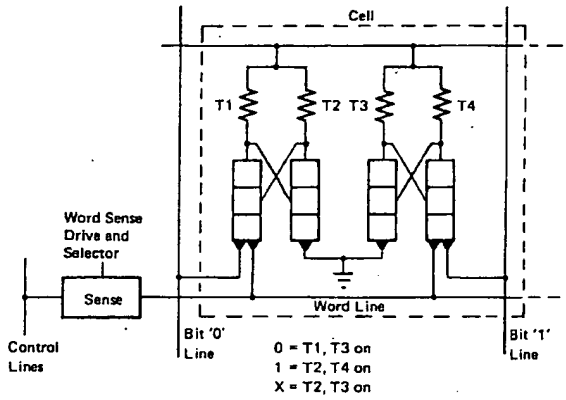

Figure 3. FM Cell

During a Select phase the two bit lines, driven antiphase from a single input-output register position, are used to interrogate the cell. It will be seen that raising the BIT 0 line with a zero stored in the cell diverts current into the word line. Current is also diverted into the word line where the cell contains a '1' and the BIT 1 line is raised. Bit line manipulation has no effect on the word line currrent where the cell state is X.

A match condition is sensed when there is an absence of word line current during the Select phase. The convention is, therefore, that the BIT 1 line is raised when interrogating for a '0' and similarly the BIT 0 line is raised when interrogating for a '1'. A latch per word retains the result of a Search operation upon the word with which it is associated. This latch is termed a 'selector'.

The Read operation is performed from words previously selected in the Select phase. To do this the selected word lines are raised in potential. This diverts current into the bit lines which is then sensed by the bit sense/driver circuits.

The cell Write operation is performed by first raising the appropriate bit line(s) such that they represent the data to be written. When the word line, selected in the Select phase, is subsequently raised in potential the two cell halves assume the state represented by the bit lines.

Figure 4 shows an array in which a cell of the type described is situated at every intersection of bit line pairs and word lines. Each bit of the input-output register has access to its related column in the array via its respective mask gate and bit sense/drive circuit.

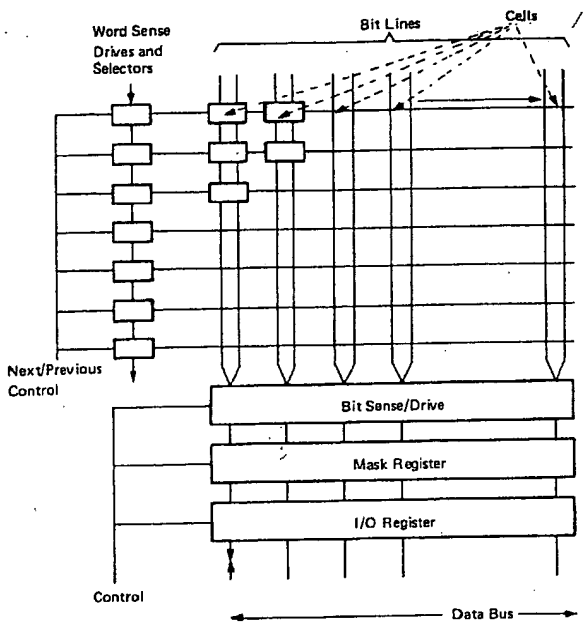

Figure 4. FM Array Layout

During a Search operation the contents of an input-output register are compared with the contents of all the array words within its BOM. The comparison will occur over the positions identified by '1s' in the mask register. In this way the cell columns over which a given Search operation is to be performed may be defined by a particular mask register pattern.

The presence of 'X' states within the array can result in a match condition existing between the contents of the input-output register and multiple words within the array. The selectors for words identified in this way are set to '1' at the end of the Select phase.

The contents of selected words only are taken into consideration when performing array Read. Using this rule the input signal to each bit in the input-output register is derived from the OR of the contents of its related cell column. The transfer from array to input-output register is gated by the mask register. In this way the bit positions, over which a read operation is performed, may be defined by a particular mask register pattern.

The Read operation senses the signals on only the BIT 1 lines and, where they are equal to '1', the corresponding input-output register bits are set. The other positions are reset. This transfer is gated to occur for positions where the mask register bit is a '1'. Because the signals on the BIT 0 lines are ignored, neither the X state nor the zero state in the cell affect the result of the Read into the input-output register.

The Write operation causes data in the input-output register to be set into the words selected during the Select phase. Array columns identified by a mask register bit equal to '1' are the only ones affected by this function.

The mask register pattern is derived from the mask store which contains patterns corresponding to the table formats within the array. This is shown in Figure 1. Separate Select phase and Read/Write phase masks are normally supplied to the mask register by a timing control within the FM BOM. The identity of the mask required is decoded by the FM BOM from a control pattern. Control patterns supplied to the FM BOM from the control source select the required masks together with the Search, Read and Write operations.

Single selector latches per array word are normally employed. In some applications a second group of selectors can provide greater logic flexibility. The selector group used in any FM cycle are defined by the control source. It is normal to link the selectors together within the FM BOM in the form of a shift register. This provides the only fixed logical link between the individual words of the array. A control function, called Next, enables a pattern of set selectors to be shifted one position through the shift register. A function similar to Next, termed Previous, provides a selector shift in the reverse direction. Using the selector shift, adjacent words to the ones currently selected can be read from, or written into, during an FM cycle. Search and either Next or Previous, specified to occur in the same cycle, perform selection of words adjacent to those which match the search pattern.

An FM store of a greater capacity than that defined by a single FM BOM can be formed by one to one linking of the input-output register connections on a number of BOM's. When a number of FM BOM's are bused together in this way the result is a store which has the same logic characteristics as a single BOM.

On completion of a cycle, an FM store will contain identical data in the input-output registers of the constituent BOM's. This is achieved in practice by first setting the individual input-output registers from the local array. Subsequent transfer to and sensing of the bus, results in the OR of the results in individual BOM's appearing in all BOM's. This bus transfer period is shown in Figure 2.

Significant improvement in utilisation of the array can be achieved by the provision of two similar input-output registers within the FM BOM. This BOM organisation is outlined in Figure 5. The contents of either of these input-output registers

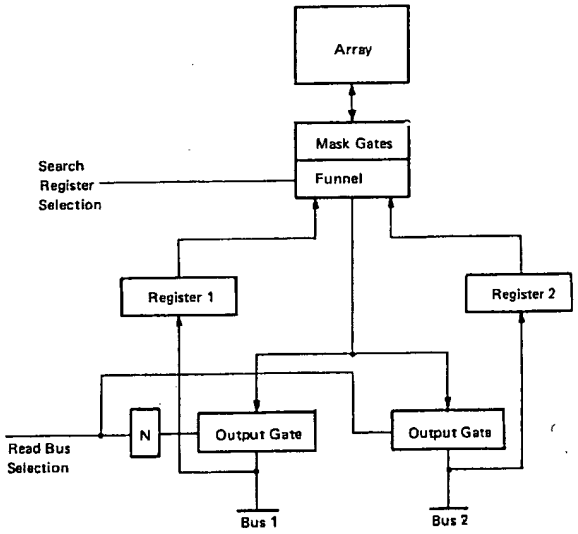

Figure 5. Two I/O Register BOM

may be specified to provide the search pattern. The output of the array may be set into either register and output onto its associated bus. Both registers are set from their respective buses.

The control sequence is as follows:

1. The register specified for the Select phase is used to generate the search pattern.
2. The array is searched and read into the register specified for the Read/Write phase.
3. The read-out from the array is output onto the bus specified for the Read/Write phase and both registers are set from their respective buses.

Note that although both registers are set from their buses only one supplies the search pattern in the next cycle. The register is specified by the control for the next cycle.

2. TABLE DESIGN

Minimising Tables

For FM to be effective as a system technology it should be able to generate any function of its inputs. One way of interpreting the array is as a three stage logic device.

The first stage (the cell) either chooses the true or inverse of the input or ignores it. The second stage (the word) is an AND of the values chosen by the cells in the word. The third stage is an OR of the selected words.

A simple analogy, seen in Figure 6, is able to generate any function of its inputs dependent on the cell states.

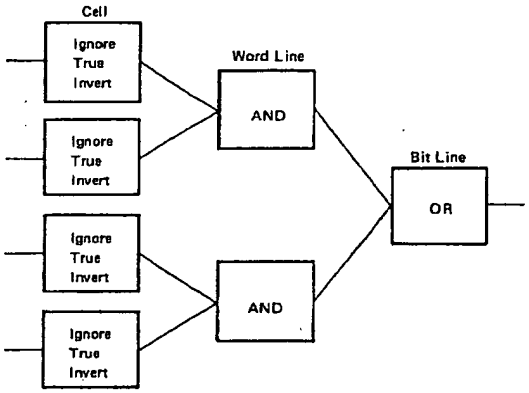

Figure 6. Logic Analogy

When the personality of a system is tabulated in storage, minimising the size of a table is an important design task. Tables are amenable to reduction by the usual logic techniques; by minimising to the disjunctive normal form of the logical expression before conversion into table form. The table in Figure 7 will generate the logical functions of two three-bit operands. The table is searched over the 10 bits of the "table name" A and B fields, and read from the 3 bit result field. The "table name" or "tag" bits are used in combination to select any of the possible 16 logical functions of the two operands. The contents of the I/O register in Figure 7 shows the search fields and the result for an XOR operation.

For clarity the figures illustrating table design use blanks to represent those cells holding the X state.

Note that this table, using FM *3-state* associative cells, requires only 12 words. The same functions, implemented in an associative array composed of *2-state* cells, would require 1024 words.

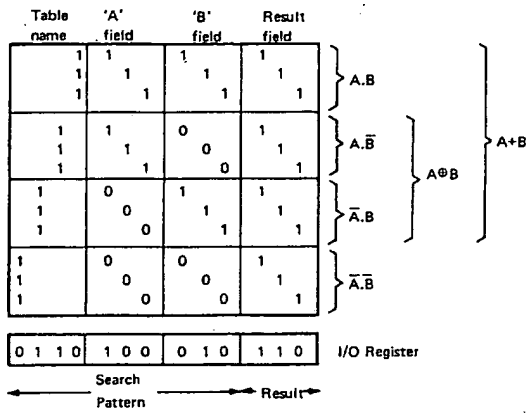

Figure 7. Complete Logic Table

Reduction techniques have been designed that are particular to FM. These are based on examining the bit patterns in FM, rather than regarding FM as an array into which Boolean expressions may be mapped. Various forms of an increment table will be used to describe these methods.

In Figure 8, four bits form the field to be incremented. The bits are referred to as S8, S4, S2 and S1 in order of decreasing significance. The next five bits form the result field C0, R8, R4, R2 and R1. This table is identical to the truth table for the same function. It performs repetitive increments inefficiently because its output is in a different field from its input. It requires the operand to be re-aligned after every increment.

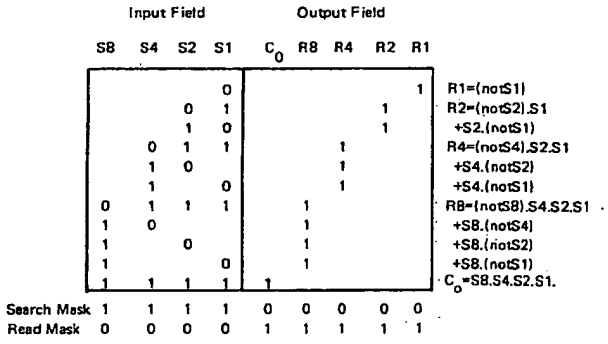

Figure 8. Search Read Increment Table

Using a Search Next operation in the Select phase, followed by a Read operation in the Read/Write phase, the increment table shown in Figure 9 can produce its output in the search operand field. This figure shows how a tag bit is introduced to prevent the matching of the search argument with words intended to output only and also to prevent the subsequent incorrect read-

ing of input words. Odd-numbered words (tagged 0) are searched for, and even numbered words (tagged 1) are read out. Note that use of the Search Next Read operation doubles the number of words in the increment table from 11 to 22 words, but almost halves the table width.

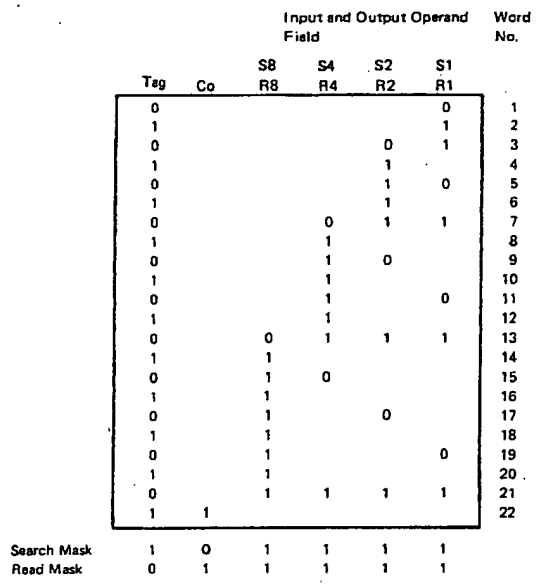

Figure 9. Search Next Read Increment Table

The number of words can be reduced, without increasing table width, by allowing words to be both searched for and read-out. For example, in Figure 9, word 3 is a searched-for word; it contains 1's in the same position as word 2, which is a read-out word. Word 2 may therefore be deleted, and word 3 used as the read-out word for word 1. Words 4, 6, 8, 10, 14, 16, 18 may be similarly removed. With these deletions, the 22 word table in Figure 9 shortens to the 14 word table shown in Figure 10.

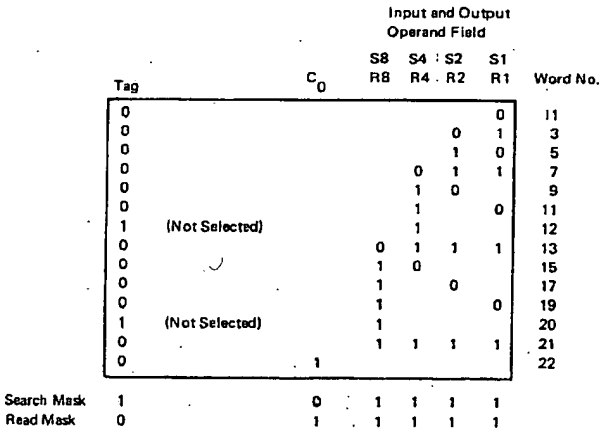

Figure 10. Minimised Increment Table

This reduction method may be applied to most Search Next Read tables, if not directly, then after word re-ordering.

Words Shared between Functions

Certain assignments of tag values allow words to be shared between functions. A simple example of this is the logic function table of Figure 7, where the 16 logic functions share the words of the table.

Any single table may be used for more than one function by searching with different masks, and by operating the FM with Search Read or Search Next Read. The advantage of sharing words, or tables, is that the number of words used is reduced.

As an example of this the table in Figure 11 may have been generated for the function 'move A to B, maintain C', where in this case A, B and C are single bit fields. The table is searched over the field 'tag AC' and is read over the field 'BC'. If the tags in this table are changed to XI and IX, as in Figure 12a, then it is possible to perform not only 'move A to B and maintain C', but also other functions, some of which are listed in Figure 12b. The function performed in any one cycle then depends upon the choice of search mask, read mask, type of search operation and value of key (input pattern used to search the tag field).

Tag	A	B	C
11	1	1	
11			1

Figure 11. Logic Table

Tag	A	B	C
X 1	1	1	
1 X			1

Figure 12(a). Improved Logic Table

Function	Search Mask	Read Mask	Key	Operation
A to B	Tag A	B	01	Search
B to A	Tag B	A	01	Search
A to C	Tag A	C	01	Search next
C to A	Tag C	A	10	Search previous
B to C	Tag B	C	01	Search next
C to B	Tag C	B	10	Search previous
B to A, C to C	Tag BC	AC	11	Search
A to B, C to C	Tag AC	BC	11	Search
A to A, C to C	Tag AC	AC	11	Search
B to B, C to C	Tag BC	BC	11	Search
A.B to A, C to C	Tag ABC	AC	11	Search
A.B to B, C to C	Tag ABC	BC	11	Search
A.B to A, A.B. to B, C to C	Tag ABC	ABC	11	Search
A.B. to C	Tag AB	C	01	Search next
C to A, C to B	Tag C	AB	10	Search previous

Figure 12(b). Logic Table Function List

Some of the above functions may Search and Read over wider fields than those shown: for example the 'A to B' function might use search mask 'tag AC', read mask 'BC', without altering the function.

In the above example a two word table was shown to provide fifteen different functions. Most tables provide a multiplicity of functions and this factor can be exploited in optimising a design.

Table design starts with a requirement for a set of functions and ends with bit patterns in several ordered consecutive words. Generation of bit patterns involves using the minimum number of adjacent words to give the required functions. The bit patterns could be generated by computer program, using as input the required combination of functions together with the FM tools available, but for some applications an ad hoc method may be found to provide better optimisation. The bit pattern structure differs in concept from the truth table interpretation that could be placed on the increment table described in Figure 8.

3. SYSTEM DESIGN

Control of FM Stores

An FM Store in a data flow is controlled by a field emitted from a store (which may be an FM Store). This field has three parts: Function, Mask and Key. These inputs are seen in Figure 1.

The most primitive FM operations are Search or Next, which are operations on the selectors, and Read or Write which are operations on the selected words.

In practice Select and Read/Write phases tend to alternate because successive primitives such as Search Search or Write Write are redundant.

It is therefore convenient to define a composite function rather than use the primitives. The composite function has two phases as shown in Figure 2. The first is the Select phase, in which Search, Next, Previous and other operations on the selectors may be specified. The second is the Read/Write phase in which the selected words are either read or written.

The advantages of the composite function are:

1. The number of control words used is halved because in the environment of table look-up data flows, a Select phase now precedes each Read/Write phase.
2. An FM store is able to control another FM store, or itself, because it allows an output as a result of each control input.
3. The transfer of data between two FM stores is simplified if they are synchronised by phase. Any speed advantage given by a sequence of primitives over a sequence of composite functions is lost in synchronisation when transferring the result to a second store.

The part of the control field specifying the composite function is typically 4 bits wide.

1. Search
2. Next
3. Previous
4. Read/Write

The first bit specifies Search and if no Search is specified then the selectors retain their setting from the preceding cycle. If a Search is specified Bits 2 and 3 define the selector shift operation which is to follow.

Possible selector shift operations are:
no shift
shift Previous
shift Next
Bit 4 specifies Read or Write.

The second part of the control field addresses the mask store. The widths of the FM store over which searching, reading or writing occur is defined by a mask. The mask address specifies two masks, one to be used during the Select phase and the other to be used during the Read/Write phase.

A field of the FM store is reserved for the tag as shown in Figure 13. Each word of the table contains the same tag value in this field. Every table must have a tag and different tables have different tag values. To select a table the store is searched using a key pattern which will match the tag for the required table. The key is supplied to the FM store from the third part of the control field.

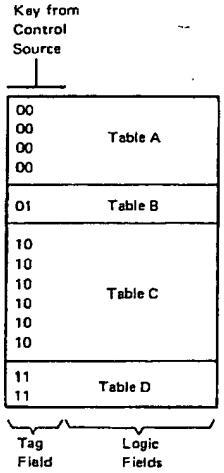

Figure 13. Table Addressing

A field of the FM bus can be returned to the control store as a micro-instruction address modifier to allow for conditional execution of microprogram (condition field). Figure 14 shows the organisation of such a control system.

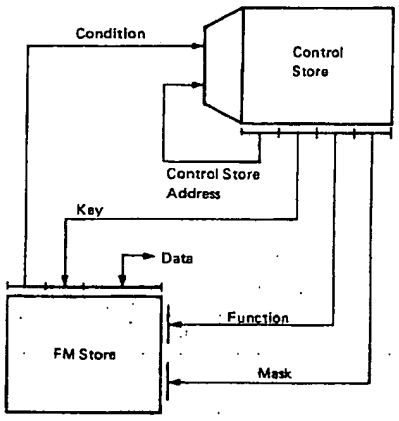

Figure 14. Control of an FM Store

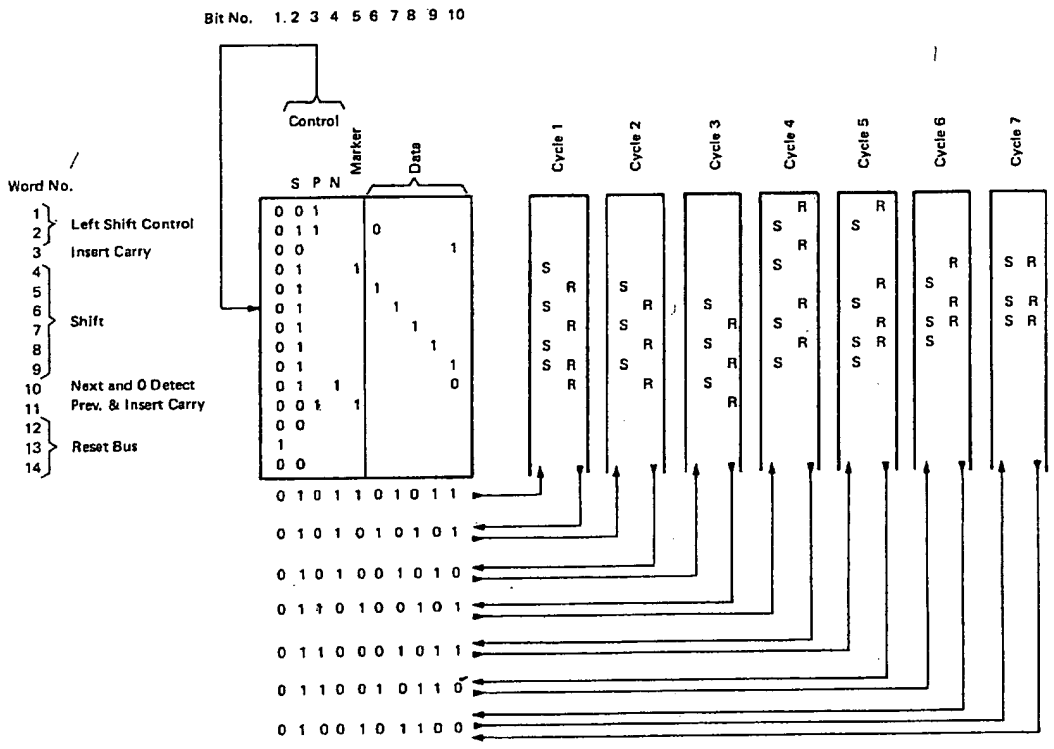

Figure 15. Auto Sequencing

Auto-Sequencing

Certain system designs require that a part of the system operates independently for a period, e.g. an I/O adapter. The principal function performed is generally simple (transfer of a data block), with intermittent need for more complex operation (test the status of the transfer). The cost of an FM based system can be reduced if those partly independent areas can generate their own control patterns for the simple sequences, and share a control for the complex sequences. The auto-generated pattern may be all, or part of, the control input to the FM store. A store, containing a table to increment by '1' the contents of a five bit field, is shown in Fig.15.

This store is able to specify its own Search, Previous and Next controls by reading onto bus bits 2,3 and 4 respectively. The table increments the five bit field defined by bits 6-10 by right shifting with the Search Next Read function. This is repeated until the least significant bit is zero, whereupon a '1' is inserted into that bit position. Then the shift reverses (using Search Previous Read), zeros being shifted in until the field is realigned. The table then maintains the result (Search Read).

Bus bit 5 is used as a marker position. It is initially set to '1' to provide a frame bit to delimit the data. It is also used to hold the carry when the shift reverses.

Words 1 and 2 control the left shift.
Word 3 places the carry in position.
Words 4 to 9 form the shift.
Work 10 generates the Next control and detects a zero in the least significant position.
Word 11 generates the Previous control and places the carry in the marker position.
Words 12, 13, 14 are used to reset the bus. A Search and Read with a '1' in bit position 1 resets the control and data fields. This operation, followed by a Search Next Read with a '1' in bit position 1, will reset the bus.

In cycle 1 of the example the external control loads the data and initiates the increment by inserting a 1 into bits 2 and 4. Thereafter the FM is auto-sequenced.

In cycles 2 and 3 the data is shifted right to shift out the least significant 0.
In cycle 4 the data is shifted left and the carry introduced.
In cycles 5 and 6 the data is realigned by shifting left.
From and including cycle 7, the data is maintained on the bus until the store is reset by external control.

This example demonstrates not only auto-sequencing but also how control has been assimilated into a table in an array.

A store can be considered to be auto-sequencing if it modifies its own control, mask or key, or any combination of these. Auto-sequencing is a technique that can be applied to more complex functions than those shown. The major limitation of auto-sequencing is the need for regeneration of control following a Write operation.

Pipeline

The execution of functions can be spread over several cycles using tables which generate components of the function. In the case of addition, the use of the factors XOR, CARRY

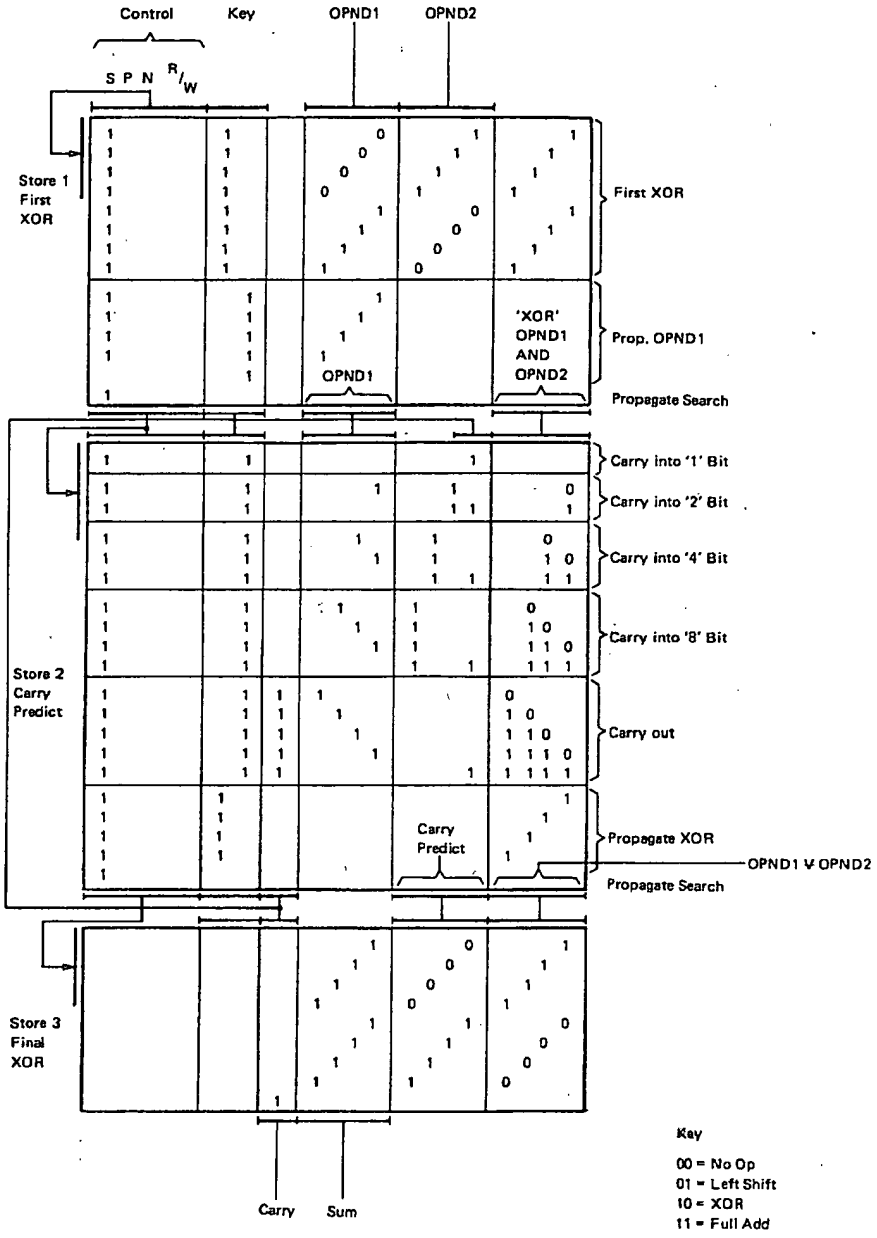

Figure 16. Pipeline

PREDICT, XOR reduces the number of words used from $2^{n+3} + 2^{n+1} - 4n - 9$ to $\frac{n^2}{2} + \frac{15n}{2} + 5$

In the case of byte addition the number of words used is reduced from 2519 to 97.

These partial function tables can either be in a single store, or in separate stores which form a pipeline. Each of these methods uses the same number of words. The pipeline scheme can give three times the throughput when fully utilised.

A three stage pipeline which performs a digit add is shown in Figure 16. Each stage is an FM store consisting of FM BOM's each having two I/O registers.

Store 1 forms the XOR of operands 1 and 2, which is passed, with operand 1, to Store 2.

Operand 1 and the XOR of Operand 1 and 2 are used by Store 2 to generate a carry predict pattern which is passed with the XOR to Store 3. An interconnection between the lower and upper I/O registers provides a path for carry propagation between digits.

Store 3 forms the sum by XORing the XOR and Carry Predict patterns. Control and key for Stores 2 and 3 are generated by the operation in Stores 1 and 2, respectively.

The tables in Stores 1 and 2 have been tagged so that the

pipeline will perform four functions. These are chosen by the keys presented to store 1:

00 NO OP
01 LEFT SHIFT OPERAND 1
10 XOR OPERAND 1 AND 2
11 ADD OPERAND 1 AND 2

This example demonstrates how stores can be interconnected to form a data flow.

CPU

Figure 17 shows how three FM stores might be interconnected to form the data flow of a CPU. Also in the diagram is a conventionally addressed Main Store and a microprogram control store.

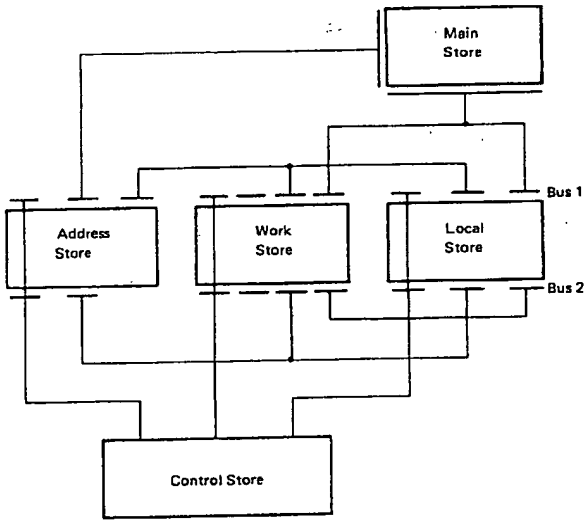

Figure 17. Data Flow

The operations performed by each FM store can be only broadly defined because, as might be expected, each store contains a mixture of tables used for logic operations, for storage and for control and decode purposes. In other words the basic functions of a data flow are much more diffused than in a conventional data flow. This can be illustrated by the way in which data patterns, exceptional conditions and decoding are recognised. In a conventional logic data flow these are very often recognised by matching data or conditions against a set of stored constants, using the arithmetic unit of the data flow to select a path through a microprogram decision tree.
Using FM it is generally cheaper and faster to form condition sensitive tables which directly detect the pattern. These tables are placed so that they operate on the data at its source in the data flow. Data need not be routed around the data flow to a single localised condition or decode unit.

The functions of the three FM stores may be broadly categorised as follows:

The Address Store provides the Main Store addressing path; it provides storage for addresses and an increment table which can be used in parallel with other arithmetic operations in the data flow.

The Work Store holds tables for basic data manipulation, for instance addition, subtraction and logical operations.

The Local Store provides working registers for temporary data storage.

It must be emphasised that these distinctions are a matter of degree. In fact all three stores perform some function of condition detection, instruction decoding, data storage and data manipulation.

4. CHECKING AND AVAILABILITY

Checking

The physical and logical characteristics of the functional memory logic element determine the checking methods used in units of which it is a part. Its structure, in which logic tables can operate correctly without being tied to specified locations in a store, suggests significant potential for greater systems availability.

It will be noted that a major feature of the array is the possibility that multiple word selection will result from a Search operation.

Normal checking methods, using parity applied to a word or field of the array, are suitable for instances of single word selection. They cannot be used where the OR of multiple words generate the result. This is due to the lack of a simple relationship between the parity of individual words and the data read from the array into the input-output register.

It is seen, therefore, that because of multiple word selection, simple parity checking methods are unsatisfactory where a three state cell is employed. An alternative way of using parity is to combine parity prediction tables and function tables within the array. This approach uses a considerable table volume within the FM stores for prediction, and requires input-output register positions to be allocated for check bits. It results in more complex designs and implies restrictions in table structure.

An alternative to the above methods is outlined in Figure 18. The FM BOM is designed with two identical arrays. The checking

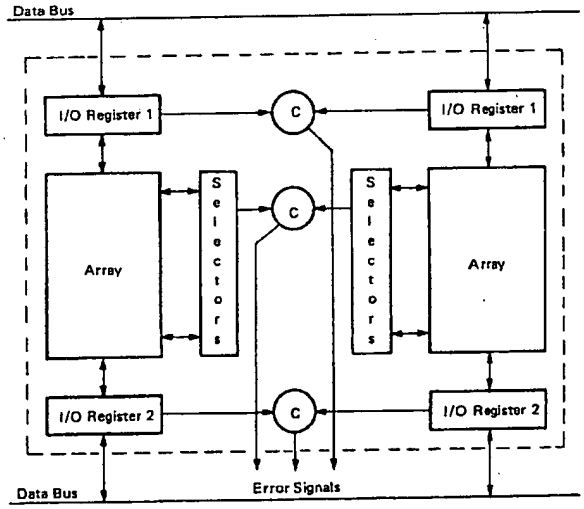

Figure 18. Checked FM BOM

function is performed by comparators (C) monitoring the selectors and input-output registers. In use the two arrays are loaded with identical tables and the comparators check each cycle for agreement. An error condition results in a compare check.

Checking by duplication in the FM BOM enables error detection to be performed independently of the function installed in the tables.

Availability

The duplication, which is included in the FM BOM for checking, can also be used to improve system availability. The checking method described can detect an error in any bit position of the data flow, but will not immediately indicate its cause. When the cause of the error can be determined there is a unique potential for self repair.

The error may be caused as a result of a transient condition within the FM data flow, which can result in spurious changes in the stored tables. Where this is the case, restoration of the tables within the arrays enables correct operation to be resumed. Diagnostic techniques can isolate the fault to an FM BOM and indicate which of the duplexed arrays contain the faulty data. The repair method is to transfer good table data from the duplicate array into the array in which the error is detected.

When the control function is a Search the correct operation of the table in the FM Store is independent of the location and ordering of the constituent words. Similarly a table employing Search Next may appear in any group of adjacent words. These properties can be used to allow the repair of permanent array failures within a store. This is achieved by transferring either words or tables from the faulty area to spare locations reserved for the purpose.

A practical implementation of this technique is shown in Figure 19. It consists of allocating a spare FM BOM to an FM store in which repair facilities are required. Persistence of an error in a specific location is normally the criterion for employing the spare FM BOM. Table data from the good half of the failing BOM is transferred to both sides of the spare BOM and it becomes an operational part of the store. The faulty component is removed logically. The Figure shows BOM's A, B and C, which form the normally used part of the store prior to error. Activation of the error line results in alteration of the pattern in the configuration register, which then indicates the new store organisation.

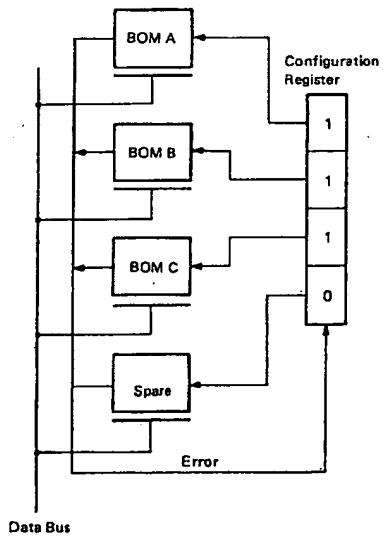

Figure 19. Repair Using a Spare BOM

CONCLUSIONS

A requirement of a system building block is generality of application. This paper is intended to show how the functional memory (FM) concept satisfies this criterion. Any evaluation of a new technology must not be based on a specific operation such as addition, but rather on the way in which it performs the total system function.

The functional memory approach introduces new standards of acceptability in which the increased number of circuits, required to implement certain system functions, must be balanced against significant benefits. These are the provision of a single module of logic which can be personalised by changeable tables, and combines a high level of checking with intrinsic repairability.

The definition of the system logic in a group of writable stores is an important departure from current practice. It provides for systems in which the logic can be implemented using many fewer part numbers than are possible today.

The function of an FM system can be modified, without physical change, by alteration of the table structure. This is useful in the design phase, where errors, and changes in system requirements, can be easily contained. It simplifies the engineering change problems in the field and gives greater scope for designs personalised to a users needs.

It is likely that with reductions in the costs of logic circuits there will be more emphasis on improving system availability. The results will be greater user satisfaction and lower maintenance costs. The capabilities of FM for performing self repair and restart are significant in this context.

Systems built in current technologies normally use diagnostic methods which are specific to each unit type. The reason is that fault location, often imprecise, requires a detailed knowledge of the logical structure being diagnosed and, further, uses this structure to assist in the diagnosis. Where logic is held in a writable array this situation changes. Testing the integrity of a logic element in a system need not now be tied to its functional use. The diagnostic procedures can be similar to those used for other memory structures, and can operate using different stored data from that required for functional operation. A benefit here is that there are now separate methods of diagnosing component failures and design faults.

The checking, by duplication built into the FM BOM, simplifies the problem of identifying a faulty module. This should reduce both equipment repair time and the degree of training required by field personnel. It should also reduce the cost of module testing during manufacture.

It is felt that current systems have been shaped by the characteristics of the available logic. This study has shown that the use of a technology with very different attributes can lead to new concepts of design. Such concepts can lead to new ways of appraising designs and the result could be a marked change in the systems of the future.

REFERENCES

1. B. T. McKeever, Proceedings – Fall Joint Computer Conference 1965.